Spécification du service actif QoSINUS

Fabien Chanussot, Pascale Primet

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

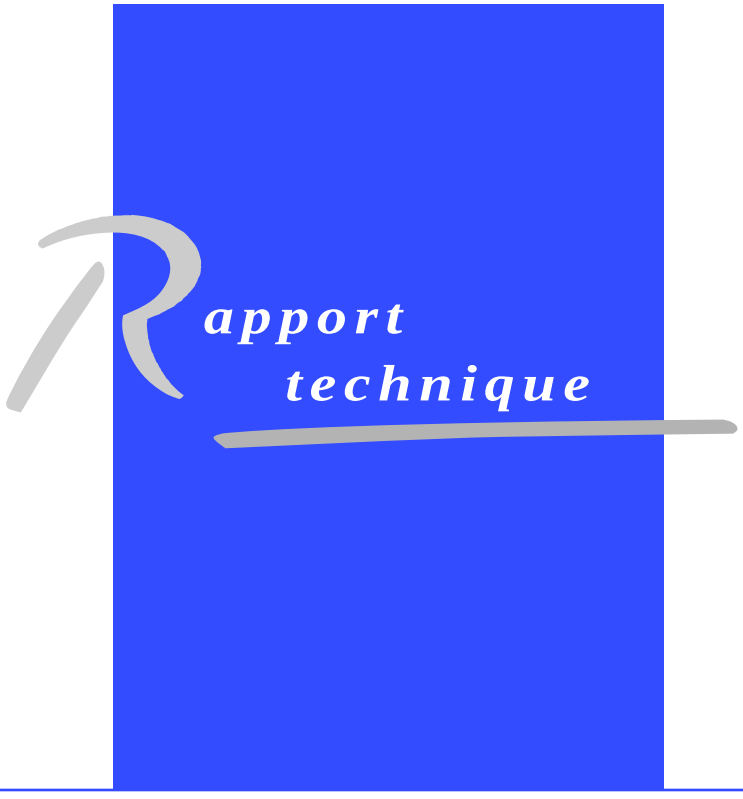
Spécification du service actif QoSINUS

Fabien Chanussot, Pascale Primet

N° 0287

7 Octobre 2003

THÈME 1



ISSN 0249-0803

Spécification du service actif QoSINUS

Fabien Chanussot, Pascale Primet*

Thème 1 — Réseaux et systèmes
Projet RESO

Rapport technique n° 0287 — 7 Octobre 2003 — 21 pages

Résumé : Ce document présente QoSINUS, un système de gestion de la qualité de service dans les réseaux DiffServ. Le système se propose de choisir à tout instant la classe de service DiffServ correspondant le mieux à une requête applicative, en fonction des performances mesurées des classes DiffServ dans le réseau.

Mots-clés : QoS, DiffServ, réseaux actifs

* RESO, INRIA

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38330 Montbonnot-St-Martin (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

QoSINUS active service specification

Abstract: This document describes QoSINUS, a QoS management system for DiffServ enabled networks. The aim of that system is to dynamically associate the best DiffServ class to an application's request, based on the DiffServ classes performances observed in the network.

Key-words: QoS, DiffServ, active networks

1 Introduction

De nos jours, beaucoup de routeurs utilisés au coeur des réseaux hautes performances assurent des services de QoS de type DiffServ [1] . C'est le cas, par exemple, de VTHD, le réseau hautes performances sur lequel est construite la grille e-Toile. Les routeurs de ce réseau proposent plusieurs classes de service différencié (DiffServ). Quatre classes sont disponibles : EF, AF/TCP, AF/UDP et BE. Les paquets de données sont marqués dans une de ces classes, et reçoivent le traitement correspondant au niveau des routeurs.

De leur côté, les applications ont besoin qu'un certain niveau de qualité de service soit assuré à leur flux de données. Mais il n'est pas nécessairement évident pour elles de traduire leurs besoins en terme de classe DiffServ affectée aux paquets composant le flux. Les performances de certaines de ces classes risquent en effet de varier dans le temps. De plus, même si la sémantique de chacune de ces classes est sensée être standardisée, leur implémentation peut varier d'un réseau à l'autre.

Il est plus intéressant pour les applications de pouvoir spécifier leurs besoins de qualité de service en terme de délai maximum, débit minimum et taux de perte maximum, chacune de ces valeurs étant quantitative ou qualitative, comme « high », « medium » ou « low ». Une application désirant transmettre un flux audio, par exemple, n'a pas d'exigences très fortes concernant le taux de perte ($<5\%$), mais a besoin d'un débit fixe et a une contrainte importante au niveau du délai de transmission et de la gigue. Un tel besoin de qualité de service peut s'exprimer de la manière suivante : « délai = « 200ms », débit = « 64 Kb/s ». Une application interactive pourra spécifier une contrainte de débit pour assurer le transfert des interactions en temps correct : "débit = 15 Kb/s".

QoSINUS permet de faire dynamiquement le lien entre une telle spécification et le traitement DiffServ des routeurs du coeur de réseau. Pour cela, un service actif est déployé aux points d'accès du domaine DiffServ. Ce service permet à une application de programmer une qualité de service sans avoir à se préoccuper de comment elle sera assurée par le réseau. Par ailleurs, ce service mesure, surveille en continu et alloue localement et dynamiquement les classes de services réseau pour servir au mieux les requêtes des flux.

Dans le cadre d'e-Toile, les clients de QoSINUS sont typiquement les applications de la grille (flux MPI par exemple) et l'allocateur (barrières de synchronisation).

2 Fonctionalité fournie

Du point de vue d'un client, l'utilisation de QoSINUS se déroule en deux phases. Lors de la première phase, le client spécifie la qualité de service requise pour son flux de données. Le système QoSINUS analyse la requête et lui attribue un code. La deuxième phase correspond à l'invocation de la qualité de service. Le client transmet ses flux de données lors d'une session identifiée par le code attribué précédement. Un fois ses données transmises, le client ferme la session. L'API décrite ci-dessous correspond à ces deux phases d'utilisation du système.

2.1 API cliente

2.1.1 Spécification simplifiée de la qualité de service requise

La fonction suivante permet à un client de spécifier la qualité de service de son flux en termes de délai, débit et perte.

errcode QoS_Set(destination, delai, debit, perte, QoSCode *)

destination : l'adresse IP et le port de la destination du flux. Paramètre en entrée.

delai : l'éxigence en terme de délai. C'est une valeur quantitative (en ms) ou qualitative (HIGH, MEDIUM, LOW). ANY si aucune garantie de délai n'est exigée. Paramètre en entrée.

debit : l'éxigence en terme de débit. C'est une valeur quantitative (en Mbps) ou qualitative (HIGH, MEDIUM, LOW). ANY si aucune garantie de débit n'est exigée. Paramètre en entrée.

perte : l'éxigence en terme de taux de perte. C'est une valeur quantitative (en garantie de taux de perte n'est exigée. Paramètre en entrée.

QoSCode : l'identifiant attribué à la session de QoS. Paramètre en sortie.

errcode : une valeur non nulle est retournée en cas d'erreur (le niveau de qualité de service ne peut pas être assuré), 0 en cas de succès.

Description : cette fonction permet de spécifier des besoins de QoS de manière simplifiée. Le degré de contrôle est par contre limité. Si la requête ne peut être assurée, cette fonction retourne une erreur. Sinon, la qualité de service requise est garantie.

2.1.2 Spécification complète de la qualité de service requise

Cette méthode de spécification est destinée aux clients désirant contrôler la QoS de manière plus fine. Elle autorise une spécification plus détaillée des besoins de

QoS, et retourne plus d'informations. Elle est basée sur l'échange de SLS (service level specification).

Une SLS est un document XML contenant les informations suivantes :

- l'identité du flux : le flux est identifié par sa source, sa destination ou éventuellement un DSCP ;
- la conformité du flux : il s'agit ici de décrir la forme que le flux s'engagera à respecter. Ce paramètre permet à QoSINUS de dimensionner les ressources à reserver pour le flux. Il correspond aux paramètres d'un token bucket ;
- le traitement des débordements : ce paramètre décrit le traitement à appliquer aux paquets non conformes. Les options possibles sont l'élimination, le lissage ou le reclassement ;
- les garanties requises : ce sont les éxigences de QoS du client, exprimées en terme de débit, délai et perte, de manière quantitative ou qualitative ;
- un planning : ce paramètre, qui décrit la date d'émission du flux et sa durée, permet à QoSINUS de construire une prévision du trafic ;

La fonction suivante permet au client de transmettre sa SLS à QoSINUS.

errcode QoS_Request(destination, SLSin, SLSout *)

destination : l'adresse IP et le port de la destination du flux. Paramètre en entrée.

SLSin : la spécification du niveau de service requis. Paramètre en entrée.*

SLSout : la spécification du niveau de service fourni. Paramètre en sortie.

errcode : une valeur non nulle est retournée en cas d'erreur (le niveau de qualité de service ne peut pas être assuré), 0 en cas de succès.

Description : cette fonction permet de soumettre une SLS pour une session de QoS. Les paramètres SLSin et SLSout sont tous les deux les représentations en mémoire d'une SLS au format XML. La SLS en entrée contient la spécification des éxigences du client. Le contenu de la SLS en sortie dépend du résultat de la fonction. Si SLSin est acceptée en l'état, SLSout contient les informations renseignées par le réseau (typiquement l'identifiant de session (DSCP) utilisé lors de la phase d'invocation). Si les exigences de SLSin ne peuvent pas être assurées, SLSout contient une proposition de qualité de service plus limitée que le réseau pourrait supporter.

2.1.3 Invocation de la qualité de service attribuée

Les deux fonctions suivantes permettent à un client d'invoquer la qualité de service que QoSINUS a attribué à son flux. Elles doivent être appelées respectivement avant et après l'émission du flux de données.

errcode QoS_Invoke(socket, QoSCode)

socket : la socket utilisée pour transmettre le flux de données. Paramètre en entrée.

QoSCode : l'identifiant de session. Paramètre en entrée.

errcode : une valeur non nulle est retournée en cas d'erreur (code incorrect ou déjà attribué), 0 en cas de succès.

Description : Cette fonction est appelée pour débuter une session de QoS. Elle programme la socket de manière à ce que les paquets émis contiennent l'identifiant de la session. L'identifiant est un DSCP (DiffServ Code Point) codé dans le champ IP TOS des paquets. Note : cette fonction permet également d'attribuer une classe de QoS prédéfinie (premium, QBSS) aux paquets, sans passer par la phase de spécification de la QoS. La valeur -1 du paramètre QoSCode permet dans ce cas de supprimer le marquage du champ IP TOS.

errcode QoS_Release(destination, QoSCode)

destination : l'adresse IP et le port de la destination du flux. Paramètre en entrée.

QoSCode : L identifiant de session. Paramètre en entrée.

errcode : Une valeur non nulle est retournée en cas d erreur (code incorrect ou non attribué), 0 en cas de succès.

Description : Cette fonction est appelée pour signifier que la session de QoS est terminée (le flux de donnée à été transmis). Le réseau, qui maintient une liste des identifiants attribués, peut attribuer l'identifiant libéré à une autre session de QoS et libèrer les ressources réservées. Note : un soft state est utilisé ; si aucune donnée n'est transmise pendant un certain temps, le réseau libère automatiquement les ressources attribuées au flux qui n'a finalement pas été émis.

2.2 Cas d'utilisation par une grille

Le dévelopement de QoSINUS s'inscrit dans celui de l'intergiciel de la grille de calcul e-Toile. Quatre types de scenarii d'utilisation de QoSINUS sont identifiés et décrits ci-dessous.

2.2.1 Scénario 0

Ce scénario est le cas limite d'utilisation de QoSINUS. Il permet à un client de spécifier directement la classe de service DiffServ à utiliser dans VTHD pour un flux de données, en court-circuitant QoSINUS. Il s'agit de spécifier directement le DSCP à utiliser à l'aide de la fonction QoS_Invoke décrite ci-dessus.

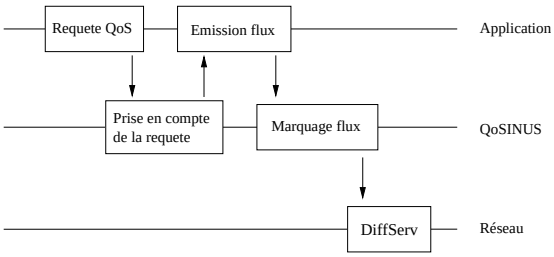


FIG. 1 – Séquence du scénario 1

2.2.2 Scénario 1

Ce scénario correspond au cas où la requête de QoS provient directement de l'émetteur du flux. Ce scénario correspond par exemple au cas d'une application classique hors du contexte de grille ou d'un flux de donnée spécifique à une application d'e-Toile qui ne mette pas en cause l'intergiciel de la grille. Il est possible d'imaginer que c'est encore le scénario qui s'applique si l'intergiciel convient d'une qualité de service prédéfinie à attribuer aux différents flux de contrôle de la grille. Un niveau de QoS prédéfini peut par exemple être spécifié par l'IHM pour les flux de donnée correspondant à une demande d'affichage de statistique d'utilisation de la grille. La figure 1 décrit la séquence correspondant à ce scénario.

2.2.3 Scénario 2

L'utilisateur de la grille peut être directement à l'origine de la requête de QoS. Un utilisateur peut, par exemple, demander à ce que les fichiers générés par l'execution d'un travail sur la grille lui soient transfèrés avec un certain débit. Dans ce cas, la requête est intègrée dans le LDT décrivant le travail. Elle doit ensuite être extraite par l'IHM, comme le montre la figure 2.

2.2.4 Scénario 3

Ce dernier scénario est celui qui correspond le plus à une application de grille. C'est le cas où la requête de QoS est émise par l'allocateur de la grille. Ce type d'utilisation de QoSINUS permet de mettre en place des politiques d'attribution des ressources du réseau. Le réseau est alors considèré comme une ressource du même niveau qu'un élément de calcul ou de stockage, et pourrait être géré par l'allocateur via son système de crédits.

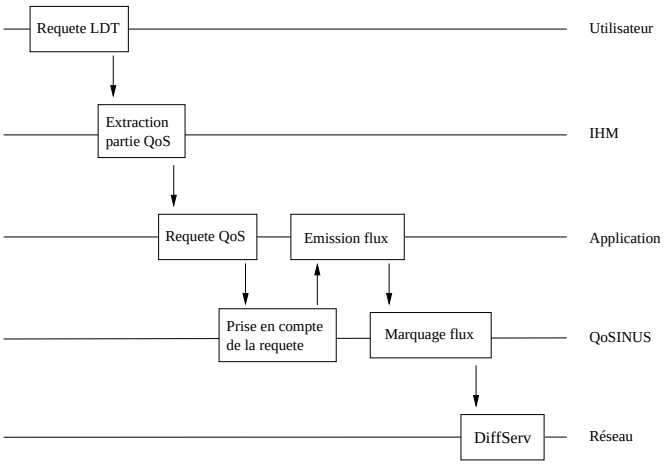


FIG. 2 – Séquence du scénario 2

QoSINUS doit dans ce cas fournir à l'allocateur une estimation des performances des différentes classes DiffServ et du coût de transport. L'allocateur utilise cette information pour établir la liste des sites à utiliser pour éxecuter un travail. Il décide du niveau de QoS à assurer aux flux du chargeur (transfert des binaires) et de l'application, et leur transmet la requête correspondante.

La spécification d'un planning dans le SLS pourrait être particulièrement utile si l'allocateur gère un planning d'éxecution des travaux, comme c'est enviseagé.

Finalement, ce cas peut permettre à l'utilisateur de fournir à l'allocateur des contraintes sur le réseau au même titre qu'il lui fourni des contraintes sur les sites de calcul à choisir.

La figure 3 expose la séquence de ce dernier scénario.

3 Architecture

3.1 Diagramme de classe du service actif

Le diagramme de classe de la figure 4 présente les éléments les plus importants du service actif QoSINUS. Le service actif QoSINUS (QaS) est la partie du système localisée sur les routeurs actifs de bordure de domaine DiffServ.

QaSEngine est au coeur du système. Il s'agit du boucle infinie qui réagit à trois types d'événements : l'arrivée d'une nouvelle requete de QoS, celle d'une demande de libération de ressources, et un pulse timer. Le timer est utilisé pour

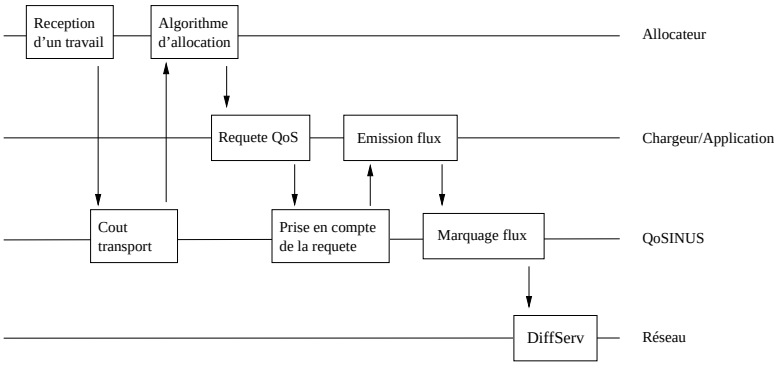


FIG. 3 – Séquence du scénario 3

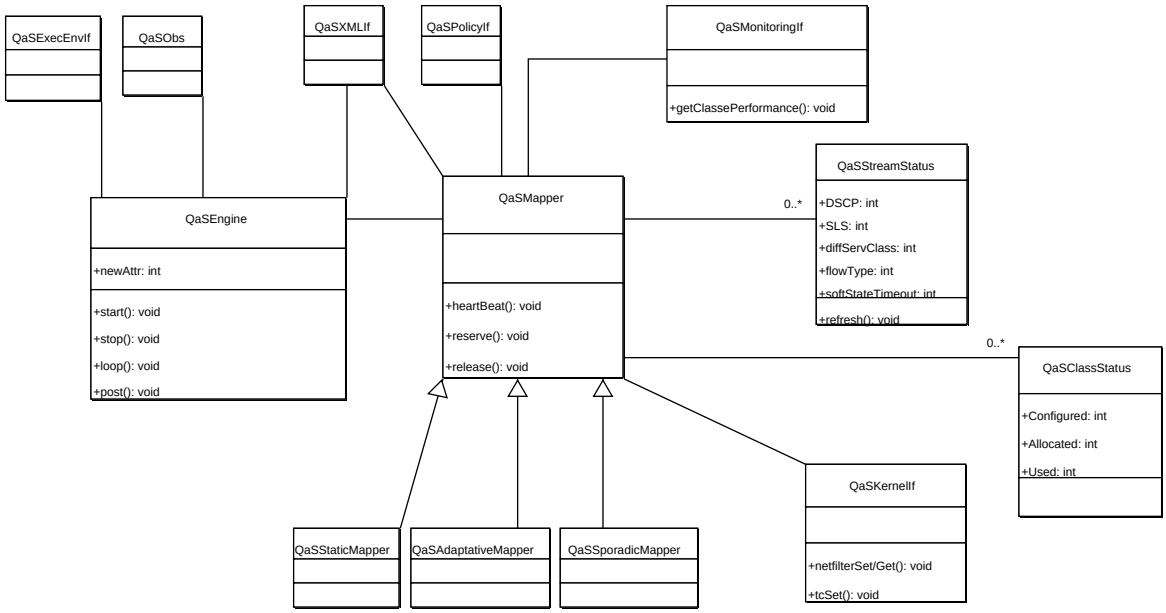


FIG. 4 – Diagramme de classe

libèrer les ressources allouées à un flux inactif depuis trop longtemps (système de "soft state" robuste en cas de mort prématurée du client).

QaSExecEnvIf fait l'interface avec l'environnement actif Tamanoir, lorsque des paquets actifs sont émis ou reçus.

QaSMapper est l'algorithme qui fait correspondre une classe DiffServ dans le coeur de réseau à une requete de QoS du client. QaSStaticMapper, QaSAdaptativeMapper, QaSSporadicMapper sont différentes implémentations de cette algorithme.

QaSStreamStatus maintient la liste des flux gérés à un instant donné, avec pour chaque flux, son identifiant, la SLS décrivant la requête de QoS associée, la classe DiffServ utilisée dans le coeur de réseau et un compteur utilisé pour détecter l'inactivité du flux.

QaSClassStatus groupe les informations relatives aux classes DiffServ du coeur de réseau, avec pour chaque classe, la bande passante reservée à la classe et la partie allouée de cette bande passante.

QaSKernelIf fait l'interface avec les modules Netfilter et Tc du noyau Linux. Ces modules permettent de marquer le champ DSCP des paquets et de limiter la bande passante utilisée par un flux (via un token bucket), en fonction des décisions de QaSMapper.

QasMonitoringIf permet à QaSMapper de recueillir des informations de monitoring relatives aux classes DiffServ dans le coeur de réseau.

QaSPolicy est un point d'entrée dans le système qui permet à un système de gestion de politique (l'allocateur de la grille ?) d'accepter ou de refuser une requête de QoS.

QaSXMLIf permet de parser les SLS au format XML.

QaSObsIf permet l'interface avec un système d'observabilité comme MapCenter.

3.2 Diagramme de séquence

Les diagrammes suivants illustrent le traitement au niveau du routeur actif des évènements suivants :

- requete de QoS ;
- demande de libération des ressources ;
- pulse timer ;
- reception d'un paquet de données.

3.2.1 Requête de QoS

Lorsqu'une application émet une requête de QoS pour un flux, un paquet actif est envoyé et est aiguillé, via l'environnement Tamanoir, vers le service actif QoSINUS.

L'objet QaSEngine, identifie le message comme une nouvelle requête de QoS et la transmet à QaSMapper.

Celui-ci choisit alors la classe DiffServ qui correspond le mieux à la requête en fonction d'information de monitoring et d'une éventuelle politique de gestion de la QoS.

QaSMapper sauvegarde ensuite des informations relatives au flux de données applicatif (QaSStreamStatus), et mets à jour celle concernant la classe DiffServ choisie (QaSClassStatus).

Il programme les modules du noyau Linux qui assurent le marquage des paquets du flux et le contrôle de trafic.

Finalement, un identifiant de flux est assigné et retourné à l'application.

La figure 5 illustre cette séquence.

3.2.2 Demande de libération des ressources

Lorsqu'une demande applicative de libération de ressources affectées à un flux est reçue, QaSMapper est chargé d'annuler la programmation des modules noyaux pour ce flux, et de mettre à jour les informations relatives aux flux et aux classes DiffServ.

La figure 6 illustre cette séquence.

3.2.3 Pulse timer

Un timer est utilisé pour réaliser des traitements périodiques.

Lorsque qu'un pulse timer est reçu, QaSMapper vérifie l'activité des flux auprès du module noyau Linux Netfilter. Si un flux est inactif, les ressources qui lui ont été allouées sont libérées.

QaSMapper vérifie également que les performances d'une classe DiffServ associée à un flux permettent toujours d'assurer la QoS requise pour ce flux, et change éventuellement de classe si ce n'est pas le cas.

La figure 7 illustre cette séquence.

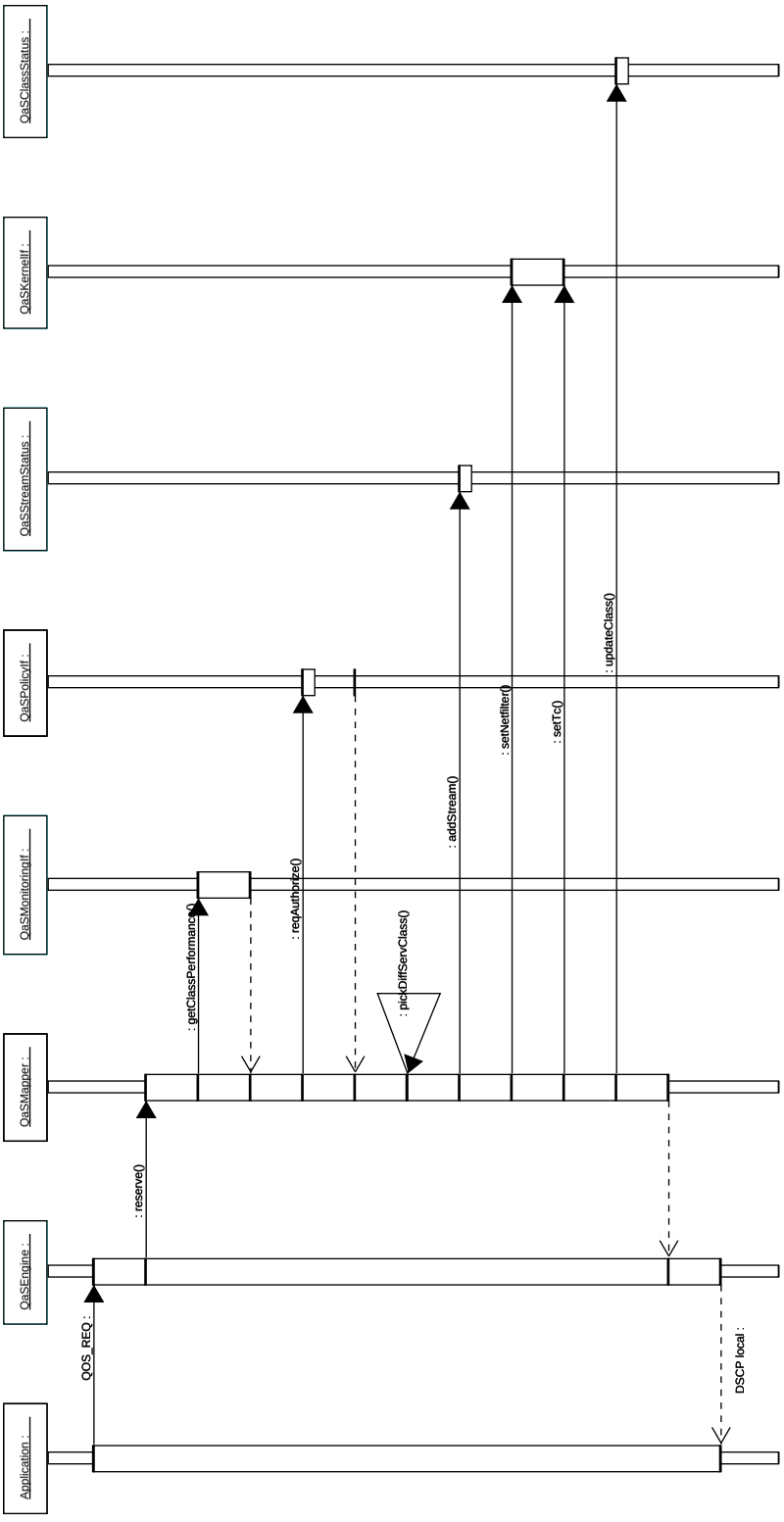


FIG. 5 – Requête de QoS

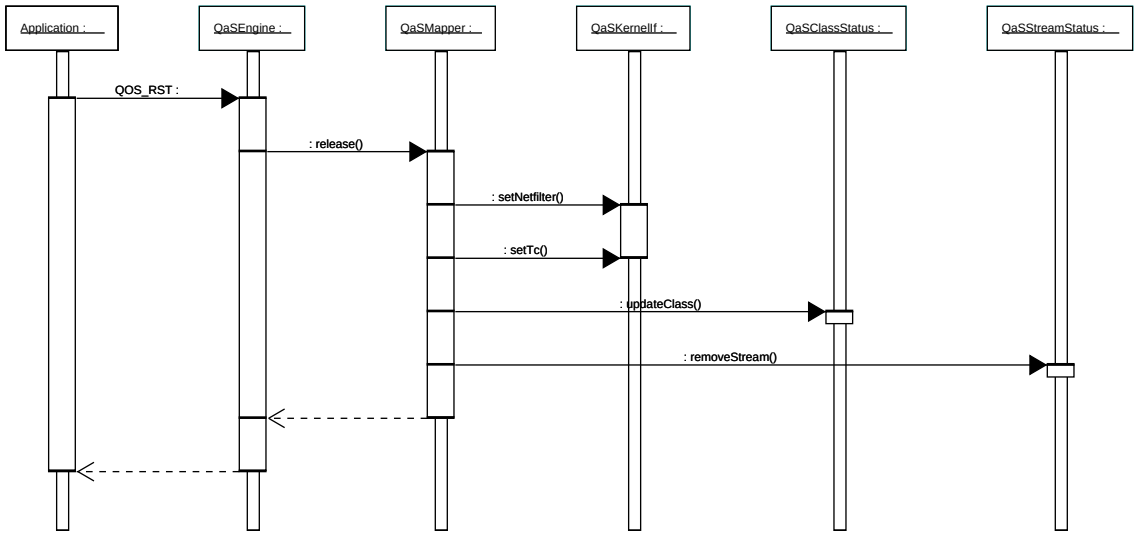


FIG. 6 – Libération des ressources

3.2.4 Réception d'un paquet de données

Lorsqu'un paquet d'un flux de donné géré par le système est reçu, un module Netfilter note l'activité du flux et marque le paquet dans la classe DiffServ attribuée au flux.

Finalement, un module Tc ("queueing discipline HTB") assure le controle d'admission du flux.

La figure 8 illustre cette séquence.

4 Implémentation

4.1 Algorithme de mapping static pour VTHD

Une première implémentation du système est disponible. Elle est basée sur un mapping static simple des requêtes de QoS sur les classes DiffServ. Elle est adaptée à l'implémentation des classes DiffServ dans le réseau expérimental haut débit VTHD.

Les routeurs actifs sont disposés au points d'accès de ce réseau. Le principe consiste, à chaque point d"accès à VTHD, à répartir la bande passante du lien d'accès entre les classes DiffServ disponibles dans VTHD. Le pourcentage de la bande

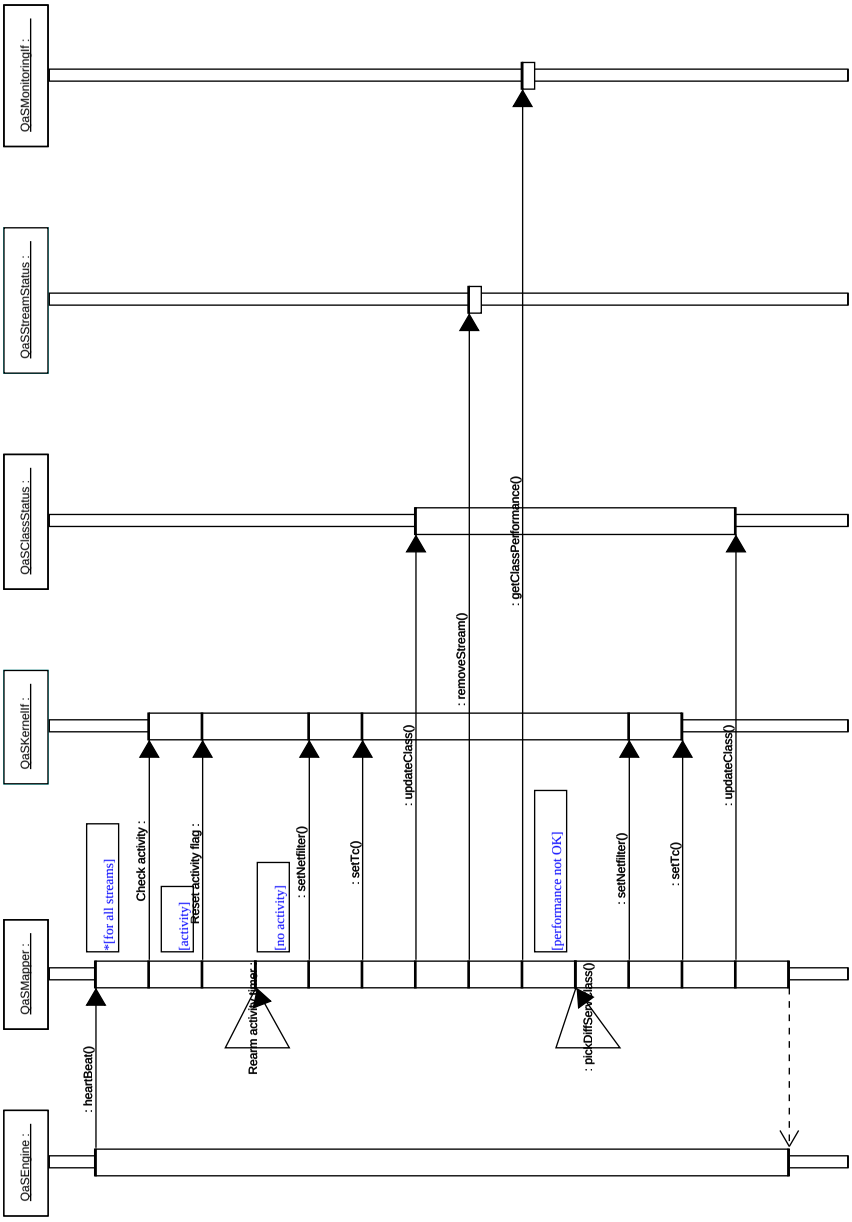


FIG. 7 – Pulse timer

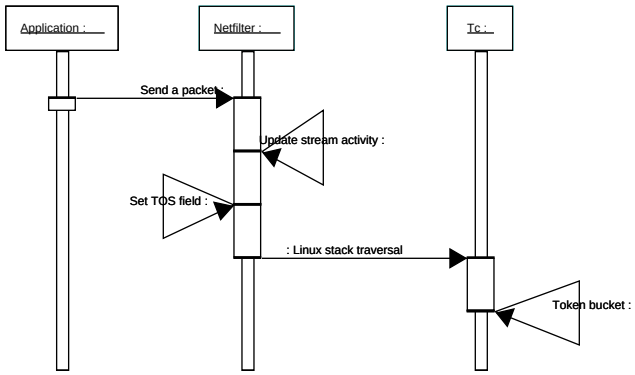


FIG. 8 – Réception d'un paquet

passante attribué à chaque classe est compatible avec la configuration DiffServ du coeur de réseau.

Lorsqu'une requête de QoS est reçue la classe DiffServ est choisie selon l'algorithme suivant :

```
si le delai demandé est inférieur à une certaine limite
alors
     si le débit demandé est inférieur au débit disponible de EF
     alors
          utiliser EF
     sinon
          si le débit demandé est inférieur au débit disponible
          de AF (TCP ou UDP en fonction du flux)
          alors
               utiliser AF (TCP ou UDP en fonction du flux)
          sinon
               utiliser Best Effort
          fin si
     fin si
sinon
     si le débit demandé est inférieur au débit disponible
     de AF (TCP ou UDP en fonction du flux)
     alors
          utiliser AF (TCP ou UDP en fonction du flux)
```

```
     sinon
          utiliser Best Effort
     fin si
fin si
```

Une fois la classe DiffServ choisie, la bande passante disponible pour cette classe est mise à jour.

4.2 Configuration du noeud actif

Un fichier de configuration regroupe, sur chaque noeud actif, les informations suivantes :

- la bande passante totale du lien ;
- pour chaque classe DiffServ du coeur de réseau, le pourcentage de la bande passante du lien réservée pour la classe, et la durée maximale pendant laquelle un flux peut ne pas emettre avant d'être déclaré inactif ;
- une table de correspondance associant à chaque valeur qualitative de délai, débit et perte, une valeur quantitative ;
- un delai limite en dessous duquel la classe EF est attribuée à une requete ;
- la valeur par défaut du débit alloué si celui-ci n'est pas spécifié dans une requête.

5 Conclusions

Le système proposé permet à une application de spécifier un niveau de qualité de service en terme de débit, délai et perte, de manière quantitative ou qualitative. Il mesure ensuite les performances des classes DiffServ du réseau, et associe à la requête applicative la classe DiffServ la mieux adaptée.

L'architecture mise en place est suffisamment flexible (échange de message au format XML, surcharge des méthodes de l'algorithme de mapping) pour permettre d'implanter rapidement de nouveaux algorithmes de choix de classe DiffServ.

QoSINUS établi ainsi en cadre dans lequel s'inscriront les développements de nouveaux algorithmes de gestion de la QoS.

6 Annexes

6.1 Schema XML d'un SLS

Les messages échangés entre l'application et le service actif QoSINUS pour spécifier le niveau de qualité de service requis sont au format XML.

Ils respectent le schéma défini ci-dessous.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<!-- XML schema that describes instances of SLS documents used to
specify QoS requirements in the QoSINUS system -->

<!--              -->
<!-- Root element -->
<!--              -->
<xsd:element name="sls" type="slsType"/>

<xsd:complexType name="slsType">
<xsd:attribute name="type" type="typeType" use="optional"/>
<xsd:sequence>
<xsd:element name="flowid" type="flowidType" minOccurs="1" maxOccurs="1"/>
<xsd:element name="conformance" type="conformanceType" minOccurs="0" maxOccurs="1"/>
<xsd:element name="excess" type="excessType" minOccurs="0" maxOccurs="1"/>
<xsd:element name="guarantees" type="guaranteesType" minOccurs="0" maxOccurs="1"/>
<xsd:element name="schedule" type="scheduleType" minOccurs="0" maxOccurs="1"/>
</xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="typeType">
<xsd:restriction base="xsd:string">
<xsd:enumeration: value="QOS_REQ"/>
<xsd:enumeration: value="QOS_ACK"/>
<xsd:enumeration: value="QOS_RST"/>
</xsd:restriction>
</xsd:simpleType>
```

```
<!--                      -->
<!-- Flow identification -->
<!--                      -->
<xsd:complexType name="flowidType">
<xsd:attribute name="dscp" type="xsd:unsignedByte" use="optional"/>
<xsd:attribute name="source" type="xsd:string" use="optional"/>
<xsd:attribute name="destination" type="xsd:string" use="optional"/>
<xsd:attribute name="port" type="xsd:integer" use="optional"/>
<xsd:attribute name="proto" type="protoType" use="optional"/>
</xsd:complexType>

<xsd:complexType name="protoType">
<xsd:attribute name="name" type="protoNameType" use="required"/>
</xsd:complexType>

<xsd:simpleType name="protoNameType">
<xsd:restriction base="xsd:string">
<xsd:enumeration: value="tcp"/>
<xsd:enumeration: value="udp"/>
</xsd:restriction>
</xsd:simpleType>


<!--                   -->
<!-- Flow conformance -->
<!--                   -->
<xsd:complexType name="conformanceType">
<xsd:attribute name="bucketrate" type="xsd:integer" use="required"/>
<xsd:attribute name="bucketburst" type="xsd:integer" use="required"/>
<xsd:attribute name="minsize" type="xsd:integer" use="optional"/>
<xsd:attribute name="maxsize" type="xsd:integer" use="optional"/>
</xsd:complexType>


<!--                   -->
<!-- Excess treatment -->
```

```
<!--                  -->
<xsd:complexType name="excessType">
<xsd:union memberTypes="dropType shapeType remarkType"/>
</xsd:complexType>

<xsd:simpleType name="dropType">
<xsd:restriction base="xsd:string">
<xsd:enumeration: value="high"/>
<xsd:enumeration: value="medium"/>
<xsd:enumeration: value="low"/>
</xsd:restriction>
</xsd:simpleType>

<!--                    -->
<!-- Guarantees ensured -->
<!--                    -->
<xsd:complexType name="guaranteesType">
<xsd:sequence>
<xsd:element name="delay" type="delayType" minOccurs="0" maxOccurs="1"/>
<xsd:element name="loss" type="lossType" minOccurs="0" maxOccurs="1"/>
<xsd:element name="rate" type="rateType" minOccurs="0" maxOccurs="1"/>
</xsd:sequence>
</xsd:complexType>

<!-- Quantitative values -->
<xsd:simpleType name="quantitativeType">
<xsd:restriction base="xsd:string">
<xsd:enumeration: value="high"/>
<xsd:enumeration: value="medium"/>
<xsd:enumeration: value="low"/>
</xsd:restriction>
</xsd:simpleType>

<!-- SLS values: quantitative or qualitative values -->
<xsd:simpleType name="slsValueType">
<xsd:union memberTypes="quantitativeType xsd:interger"/>
</xsd:simpleType>
```

```
<!-- Delay request -->
<xsd:complexType name="delayType">
<xsd:attribute name="value" type="slsValueType" />
<xsd:attribute name="period" type="xsd:integer" use="optional"/>
</xsd:complexType>

<!-- Loss request -->
<xsd:complexType name="lossType">
<xsd:attribute name="value" type="slsValueType" />
<xsd:attribute name="period" type="xsd:integer" use="optional"/>
</xsd:complexType>

<!-- Rate request -->
<xsd:complexType name="rateType">
<xsd:attribute name="value" type="slsValueType" />
<xsd:attribute name="period" type="xsd:integer" use="optional"/>
</xsd:complexType>


<!--                    -->
<!-- Emission schedule -->
<!--                    -->
<xsd:complexType name="scheduleType">
</xsd:complexType>
<xsd:sequence>
<xsd:element name="timeslot" minOccurs="1" maxOccurs="unbounded"/>
<xsd:complexType>
<xsd:attribute name="start" type="xsd:dateTime" />
<xsd:attribute name="end" type="xsd:dateTime" />
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:schema>
```

Références

[1] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. "an architecture for differentiated services", rfc 2475. Technical report, 1998.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)

Unité de recherche INRIA Futurs : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803